FINITE ELEMENT MESH SIZING FOR SURFACES USING SKELETON

William Roshan Quadros[+], Steven James Owen*, Mike Brewer*, Kenji Shimada[+]

[+]*Dept. of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA, 15213, USA.*

**Sandia National Laboratories[1], Albuquerque, NM, 871850, USA.*

ABSTRACT

The finite element (FE) mesh sizing has great influence on computational time, memory usage, and accuracy of FE analysis. Based on a systematic in-depth study of the geometric complexity of a set of connected surfaces, a computational procedure for the generation of FE mesh sizing function is proposed. The computational procedure has three main steps: (1) Generation of source points that determine the size and gradient at certain points on the surface; (2) generation of an octree lattice for storing the sizing function; and (3) interpolation of mesh size on the lattice. The source points are generated automatically using a set of tools that are sufficient to completely measure the geometric complexity of surfaces. A disconnected skeleton of the input surface is generated, and it is then used as one of the tools to measure the proximity between curves and vertices that form the boundary of a surface. Octree lattice is used as it reduces the time for calculating the mesh size at a point during meshing. The size at the octree lattice-nodes is calculated by interpolating the size of the source points. The computational procedure is independent of the meshing algorithm; it can handle non-geometric factors, and it is capable of generating variety of meshes by providing the user with enough control of mesh size and gradation. The proposed approach has been tested on many industrial models, and graded surface meshes have been generated successively.

Keywords: Surface meshing, finite element mesh sizing function, skeleton, and medial axis transform

1. INTRODUCTION

This paper examines element sizing of surface meshes used in the finite element method (FEM). The FEM is a versatile and powerful numerical procedure which analyzes complex structures and continua for various scientific and engineering fields. Surface meshes are used in industry in such diverse applications as animation, cinematography, medical simulations, manufacturing, and FE analysis. Each area of application has a specialized requirement for element quality, sizing, approximation accuracy, and computational time of the surface mesh. For example, in the computer graphics community, the emphasis is on visual quality, optimizing the number of mesh elements or polygons, and speed. Whereas, in engineering, the requirement is for element size, quality, number, and orientation. This paper will focus on automatically generating sizing information for FE surface meshing.

Even though much research has been done developing automatic, unstructured FE surface meshing algorithms [1], these algorithms do not recognize the complexity of geometry upfront; therefore it is difficult to generate the optimal mesh in one step. For this reason it is worthwhile to split the surface meshing process into two steps: (1) Analyze the input surface and generate functions that provide size, shape and orientation of the desired elements. (2) Generate FE mesh using the size, shape and orientation information. Here the objective is to completely analyze the input surface and to provide the mesh sizing function to the surface meshing algorithms.

In general, a FE mesh sizing function depends on various factors such as, geometric complexity of the domain, physics of the problem, boundary conditions etc. As geometry of the domain is most influential factor and is always available upfront, here we generate mesh sizing function for generating an initial geometry-adaptive mesh.

As illustrated in Figure 1, a geometry-adaptive mesh depends on the geometric complexity of the surface and contains significantly fewer elements, while maintaining the mesh quality with fine elements at small features and a high curvature region, and by a smooth transition in mesh size; in this way, geometry-adaptive mesh reduces computation time and memory usage during preliminary analysis without sacrificing accuracy. The accuracy of analysis can be improved later by refining and coarsening the initial geometry-adaptive mesh, based on error estimation from the preliminary analysis.

Thus generation of an appropriate geometry-adaptive mesh sizing function is crucial in obtaining accurate preliminary FE analysis results; so there is a great demand for automatic generation of a geometry-adaptive mesh sizing function.

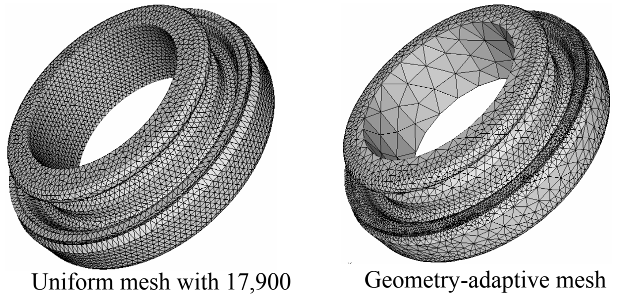

Uniform mesh with 17,900 tri elements
Geometry-adaptive mesh with 9,001 tri elements

Figure 1 Uniform mesh and geometry-adaptive mesh

[1] Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000

2. LITERATURE REVIEW

This section discusses the previous surface meshing approaches with more emphasis on sizing. Many of the previous meshing algorithms discuss some sort of element/nodal spacing control; however, sizing is integrated with the meshing process.

In early meshing schemes, users were required to specify the size manually; even today, while meshing the complex parts, user input is required in many commercial packages. In early advancing front methods (AFM) [2], a background mesh consisting of simplicial elements (triangles), was manually constructed using sample points (nodes). The size of the background mesh at each node is specified by the user, and during meshing, the size at a point in the 2D domain is calculated by interpolating the sizes stored at the nodes of the triangle containing that point. In general, specifying the size manually is a tedious and time consuming process.

The generation of background mesh was later automated by generating a Constrained Delaunay Triangulation (CDT) [3, 4] of a set of vertices. Delaunay mesh [5] on curved surfaces was later developed by satisfying empty circumellipse, rather than an empty circumcircle in parametric space. Cunha et al. [6] automated the placement of the background mesh nodes on the curves, followed by placement on surfaces, using curvature and proximity. Proximity is determined based on the distance between the facets and the nodes. Measuring proximity in this manner is a combinatorial problem and is generally time consuming. Also, because the empty Delaunay meshes are coarse, abrupt variations can occur in the target mesh size.

Owen and Saigal [7] used a natural-neighbor interpolation method to alleviate the abrupt variations in target mesh size. Owen and Saigal [7] generated a CDT in 2D parametric space by considering a sparse set of nodes on the boundary of the surface. The size at the boundary nodes is calculated based on the edge lengths of the boundary segments. The final sizing function depends on node placement in the initial background mesh, and on the interpolation method. The natural-neighbor interpolation method using CDT, has shown good results in surface meshing; however, initial distribution of boundary vertices is crucial in obtaining CDT and proper initial size at the nodes.

One disadvantage of the background mesh is that, while calculating the mesh size at a point, finding the triangle containing that point is expensive, and a parallel-developed, alternative to store mesh size is the background grid. Pirzadeh [8] used the uniform Cartesian grid to store the mesh size; however, a uniform grid is not suitable for capturing the large gradient in mesh size and it consumes huge amounts of memory. The other class of background grids which overcome uniformed grid limitations are non-uniform hierarchical grids called Quadtree.

The Quadtree, a spatial decomposition method, was pioneered for meshing in 1980s by Yerry and Shepard [9] and surveyed by Tracker[10] and Shepard[11]. The size of the quadtree cells (squares), depends on the subdivision of the bounding box, which is governed by the user-supplied spacing function or a balance condition for the tree. The drawback of this approach is that Quadtree is orientation sensitive and it is difficult to control the sizing gradient.

Another class of meshing approach uses medial axis transform (MAT)[12], for geometry-adaptive meshing. Srinivasan et al. [13] used the radius function of MAT to control nodal spacing on the boundary and interior of a 2D domain while generating adaptive triangular mesh. Gursoy and Patrikalakis [14, 15], used MAT to detect constrictions, extract holes, and to generate adaptive triangular meshes. The author has used medial axis to generate adaptive quadrilateral meshes on surfaces by varying the width of the tracks using radius function [16]. Even though MAT has been used in mesh generation, no specific research has been done in generating the mesh sizing function.

Although the approaches discussed above are effective and useful in many aspects, no serious attempt has been made to systematically understand the geometric complexity of the surface with reference to FE mesh sizing, and no general framework specifically for sizing function generation is proposed; these two issues are addressed in this paper. Sections 4 and 5 discuss understanding and measuring geometric complexity of surfaces in reference to FE meshing. Skeleton is proposed as one of the tools to accurately measure proximity and feature size. The Skeleton avoids finding distances between combinations of geometric entities. In Section 6, an overview of the computational procedure for sizing is given. The details of the general framework are given in Sections 7, 8, and 9. The proposed framework is independent of the meshing algorithm, general enough to handle non-geometric factors, and capable of generating variety of meshes. It is also computationally efficient, robust, and easy to implement.

3. PROBLEM STATEMENT

The goal of our work is to develop a computational procedure for mesh sizing function by completely measuring the geometric complexity of surfaces. The mesh sizing function must meet certain requirements: (1) Mesh size should be bounded with in a minimum size (d_{min}) and a maximum size (d_{max}). (2) The gradients should be bounded by a predefined limit (α). A more formal statement is given below.

Given a set of connected surfaces **F** *in* R^3 *and the bounds of mesh size* d_{min} *and* d_{max}, *Generate the mesh sizing function s, based on the geometric complexity of* **F**, *such that,*

1. *Mesh size* $d = s(\mathbf{p})$ *where point* $\mathbf{p}(x,y,z)$ *lies on* **F** *and* $d_{min} \le d \le d_{max}$
2. s *is* α*-Lipschitz, i.e., for any two points* $\mathbf{p_1}$, $\mathbf{p_2}$ *lying on* **F**

$| s(\mathbf{p}_1) - s(\mathbf{p}_2) | \le \alpha \, || \mathbf{p}_1 - \mathbf{p}_2 ||$ where α is a constant

4. GEOMETRIC COMPLEXITY OF SURFACES

As it is difficult to analyze the geometric complexity of the set of connected surfaces, **F**, at once, first the set **F** embedded in $\Re^3$ is decomposed into disjoint subsets; then the geometric complexity of each subset is analyzed considering the FE mesh generation. The surfaces are decomposed into disjoint subsets for the purpose of theoretical analysis only.

4.1. Disjoint Subsets of a Set of Connected Surfaces

Let set **F** contains L connected surfaces F_i, where i = 1, 2, L, which meet only at the boundary curves (without intersections), and each surface F_i has a interior and a boundary as given in Equation 1.

$$\mathbf{F} = \bigcup_{i=1}^{s=\mathrm{L}} F_i \text{ where } F_i = in(F_i) + bnd(F_i) \qquad \textbf{(1)}$$

Equation 1 can be rewritten as

$$\mathbf{F} = \sum_{i=1}^{i=\mathrm{L}} in(F_i) + \bigcup_{i=1}^{i=\mathrm{L}} bnd(F_i) \qquad \textbf{(2)}$$

Let the set **F** contain M curves C_j, where j = 1, 2… M.

$$\bigcup_{i=1}^{i=\mathrm{L}} bnd(F_i) = \bigcup_{j=1}^{j=\mathrm{M}} C_j \quad \text{where } C_j = in(C_j) + bnd(C_j) \qquad \textbf{(3)}$$

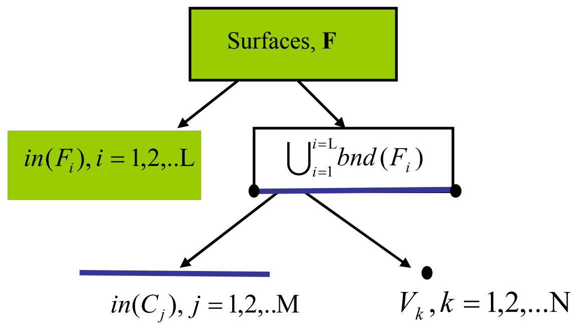
Substituting Equation (3) in Equation (2)

$$\mathbf{F} = \sum_{i=1}^{i=\mathrm{L}} in(F_i) + \sum_{j=1}^{j=\mathrm{M}} in(C_i) + \bigcup_{j=1}^{j=\mathrm{M}} bnd(C_j) \qquad \textbf{(4)}$$

The union of boundary of M curves is same as set of N vertices of **F**, and so Equation 4 can be written as Equation 5.

$$\mathbf{F} = \sum_{i=1}^{i=\mathrm{L}} in(F_i) + \sum_{j=1}^{j=\mathrm{M}} in(C_i) + \sum_{k=1}^{k=\mathrm{N}} V_k \qquad \textbf{(5)}$$

Equation (5) shows that the disjoint subsets of a set of connected surfaces are: The interior of each surface, the interior of each curve and vertices. In Figure 2, the root of the tree is the set of input surfaces (a single surface is shown for clarity), and the leaf nodes are the disjoint subsets. As the subsets are disjoint, the geometric complexity of each subset is independent of the other.



Surfaces, **F**
$in(F_i), i = 1,2,..\mathrm{L}$
$\bigcup_{i=1}^{i=\mathrm{L}} bnd(F_i)$
$in(C_j), j = 1,2,..\mathrm{M}$
$V_k, k = 1,2,...\mathrm{N}$

Figure 2 Disjoint subsets of a surface

4.2. Geometric Complexity of each Disjoint Subset

The disjoint subsets of **F**, which are embedded in $\Re^3$, are defined first in order to understand their geometric complexity. A vertex V is a point in $\Re^3$, which is defined by its Cartesian coordinates V_x, V_y, and V_z. The *in(C)* is a curve in $\Re^3$ is a continuous mapping $\alpha : I \to \Re^3$ where I is interval (a, b) on real line $\Re$. To simplify the analysis, we assume *in(C)* is curvature continuous. A general parametric surface *in(F)* can be defined by a vector-valued mapping **x** from 2D parametric space D to a set of 3D coordinates.

$$\mathbf{x} : D \to F \text{ where } D \text{ is a open subset of } \Re^2.$$
$$\mathbf{x}(u,v) = (x(u,v), y(u,v), z(u,v))^{\mathrm{T}}$$

The mapping **x** is called parameterization of the surface in (F). If for every point $\mathbf{q} = \mathbf{x}(u_0, v_0)$, there exists a neighborhood $\mathrm{M_q}$, $\mathbf{q} \in \mathrm{M_q} \subset \Re^3$ and $\mathrm{N_q}$, $(u_0, v_0)^{\mathrm{T}} \in \mathrm{N_q} \subset \Re^2$ such that $\mathbf{x}$: $\mathrm{N_q} \to \mathrm{M_q} \cap F$ is a differentiable homeomorphism on $\mathrm{N_q}$, and the differential $d\mathbf{x_p}$: $\Re^2 \to \Re^3$ is one-to-one for every point $\mathbf{p} \in \mathrm{N_q}$, the surface *in(F)* itself is called "a regular parametric surface". To simplify the analysis, it is assumed that the surface F is curvature continuous, i.e., F can be parameterized locally by a mapping $\mathbf{x}(u,v) = (x(u,v), y(u,v), z(u,v))^{\mathrm{T}}$, which is a twice continuously-differentiable function. At the end of this section, a note on relaxing this assumption is given.

The geometric complexity of each disjoint subset defined above is analyzed in reference to FE meshing in the following paragraphs. A vertex V is a zero dimensional entity whose interior is same as the boundary; it is not necessary to understand its geometric complexity. To represent a vertex V in the FE mesh, an FE node must be placed.

The geometric complexity of the interior of a curve, *in(C)* (the only subset that is an 1D geometric entity), is discussed here. Figure 3(a) shows the invalid linear FE that is longer than the curve. This shows that the proximity between the end vertices of a curve should be taken into account. As *in(C)*, a 1D subset, is embedded in 3D space, *in(C)* shown in Figure 3(a) can be bent to lie on a plane, as shown in Figure 3(b). From Figure 3(b) it is clear that curviness should be taken into account while generating the linear elements, in order to better represent *in(C)*; that is, the smaller elements should be placed at the region of high curviness. Again, as *in(C)* is 1D subset, it has one more freedom to come out of the plane. Figure 3(c) shows the helix with a non-zero twist, but with the same length and curviness as that of the curve lying on the plane (twist=0). An element paved on these two curves shown in Figure 3(c) will have different deviation from the original curve.

The geometric complexity of the interior of a surface, *in(F)*, which is the only subset that is a 2D geometric entity, is discussed here. Figure 4(a) shows the invalid triangular elements that lie outside the *in(F)*. This shows that the proximity between the boundary curves and vertices should be taken into account. As *in(F)* is embedded in 3D space, it is able to bend to form a curved geometric entity. Figure 4(b) shows the invalid triangular elements which do not represent the original curved geometry well. Thus, at the interior of a surface, the curviness should also be taken into account while generating the FE meshes.

At the beginning it was assumed that curves and surfaces were curvature continuous, but there could exist cusps (such as the tip of cone) and sharp bends in the curves and surfaces where tangent and curvature vector are not well defined. These points, curves, and regions are considered to be hard points, curves, and regions. The complexity of hard points, curves, and regions can be analyzed, as discussed in analyzing vertices, interior of curves, and interior of surfaces. And, during meshing, FE nodes should be placed appropriately to better represent these hard entities. These hard entities are not addressed in the rest of the paper.

Also, the relative position of surfaces in **F**, for example, the angle between adjacent surfaces etc., are not considered here, as they do not affect surface meshing. In some applications, due to the physics involved, the relative position could play a prominent role, but is not considered here as it is outside the scope of this paper.

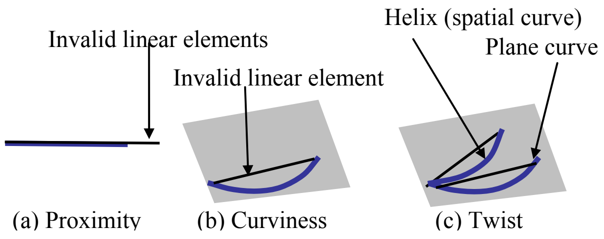


Figure 3 Geometric complexity of curve interior

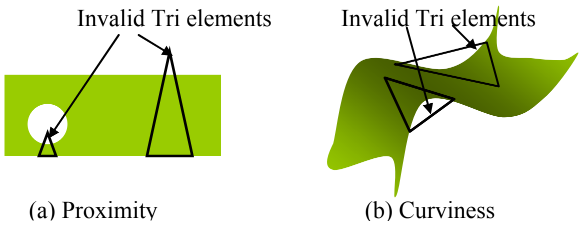


Figure 4 Geometric complexity of surface interior

5. TOOLS FOR MEASURING GEOMETRIC COMPLEXITY OF EACH SUBSET

In the following paragraphs, the tools needed to measure the geometric complexity of each subset of **F** are discussed in detail. A disconnected skeleton is proposed as the tool to measure the proximity in *in(F),* and principal normal curvatures are used as the tool to measure the curviness of *in(F)*. In *in(C),* arc length, principal normal curvatures, and torsion are used as the tools to measure proximity, curviness, and twist.

5.1. Measuring Proximity in *in(F)*

Here, a computationally-efficient, sufficiently accurate disconnected skeleton that provides local thickness information as the tool to measure the proximity in *in(F)* is proposed. Measuring proximity between the boundary entities of *F* is a global problem and the previous method employed, i.e., finding distance between all combinations of geometric entities that form the boundary, is expensive and less accurate. Therefore here the skeleton is generated using concepts of medial axis transform (MAT) and chordal axis transform (CAT).

The MAT is initially defined by Blum [12] for a planar domain as the locus of the center of the maximal ball as it rolls inside an object--along with the associated radius function (see Figure 5(a)). In other words, a point $\mathbf{q} \in F$ is contained in medial, MA(*F*), if and only if there exists a closed disc K($\mathbf{q}$, r($\mathbf{q}$)) with center $\mathbf{q}$ and radius r($\mathbf{q}$), which is not contained in a larger disc W with $K(\mathbf{q}, r(\mathbf{q})) \subset W \subset F$. The radius function r($\mathbf{q}$) provides the local thickness and is a accurate measure of proximity in *in(F)*. Blum also gave a grassfire analogy to MA, i.e., MA exists at the interior where the grassfire propagated from the boundary meet. Later, Wolter [17] defined the MAT of a closed n-dimensional topological manifold, *F,* bordered by a topological n-1-dimensional manifold, *bnd(F)*, as the subset of the cut locus $C_{bnd(F)}$, which is contained in *F*, i.e., MA(*F*) = $C_{bnd(F)} \cap F$. The cut locus $C_{bnd(F)}$ of a closed set *bnd(F)* in the Euclidean space is defined as the closure of the set containing all points, which have at least two shortest paths to *bnd(F)*. Wolter et al. [18, 19] extended the MAT on curved surfaces by finding the shortest path on the surface using geodesics distance instead of Euclidean distance. The minimal geodesic distance between two points $\mathbf{q}_1$ and $\mathbf{q}_2$, is the minimal length of all curves in *F* that join $\mathbf{q}_1$ and $\mathbf{q}_2$.

Even though the MAT is mathematically well-developed [20] and is an accurate tool for measuring proximity, it is computationally expensive to generate the continuous MAT. As MAT is defined over only continuous domains, it is generally expensive. Also, on curved surface, finding geodesic is expensive as it requires solving a system of differential equations [21].

Prasad defined a new type of skeleton for both continuous and discrete domain, called chordal axis transform (CAT)[22]. The CAT of a non-degenerate planar shape is defined as set of ordered pairs (p, δ), where p and δ are either the midpoint and half the length, respectively, of a maximal chord of tangency, or the center and radius, respectively, of a maximal disc with three maximal chords of tangency, that form an acute angled triangle. Figure 5(b) shows the chordal axis (CA), points of a discrete planar domain that can be generated with less computational cost. Note that CAT is defined only on planar surface and no effort has yet been made to define it on curved surfaces.

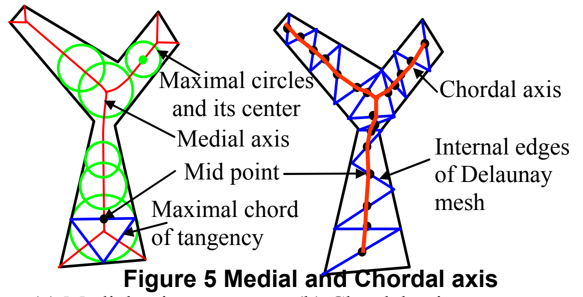


Figure 5 Medial and Chordal axis

(a) Medial axis (b) Chordal axis

5.2. Skeleton Generation

As FE mesh is a discretization of a continuous domain, in this paper, a set of sufficiently-accurate disconnected skeleton points are generated using the concepts of MAT and CAT, with less computational cost. Figure 6(a) illustrates a general surface F in $\Re^3$. First a discrete representation of parametric surface F, is obtained by extracting graphics facets, T. Note that graphics facets are generated based on the curvature of the surface and may contain poor quality triangles, as maintaining the cardinality of T minimum is one of the objectives. A set of triangles T, is split into planar (T_{pln}), and curved (T_{crv}) subsets, as illustrated in Figure 6(b). Subset T_{pln} contains triangles whose vertices lie only on *bnd(F),* and the angle between the normals of the adjacent facets t_i, $t_j \in T_{pln}$, should be within a predefined limit θ_{pln}. The subset $T_{crv} = T \backslash T_{pln}$, represents high curvature regions containing many vertices in *in(F)*. As CAT is defined on the planar domain, disconnected skeleton points are generated using CAT in T_{pln}. And in T_{crv}, grassfire is propagated from the boundary to obtain disconnected skeleton points, using the concept of MAT. The details of obtaining these skeleton points in T_{pln} and T_{crv}, are explained in the following paragraphs.

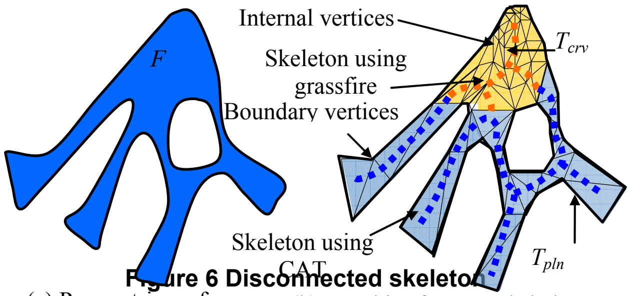
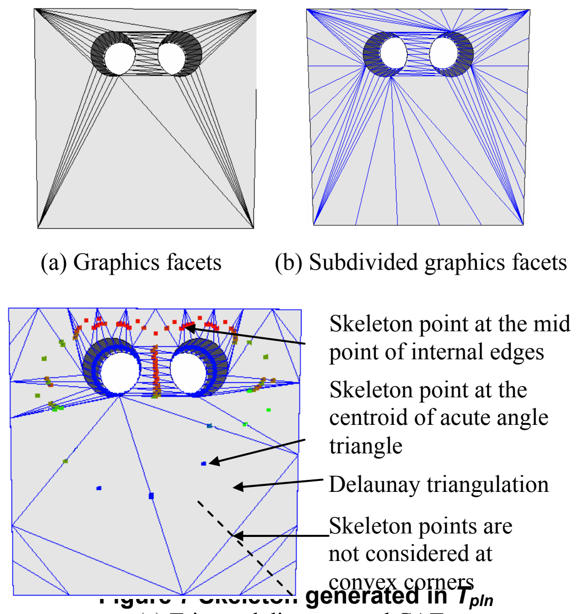


Figure 6 Disconnected skeleton

(a) Parametric surface (b) Graphics facets and skeleton

Here the disconnected skeleton generated in T_{pln}, using the CAT, is explained. T_{pln} of a surface contains fewer facets and does not guarantee quality (see Figure 7(a)). To increase the number of triangles, the triangles are subdivided by adding Steiner points only on the boundary, as shown in Figure 7(b). The addition of Steiner points depends on the number of boundary edges and the internal angle of the subdivided triangles. As Delaunay mesh is needed for the generation of CA, triangulation shown in Figure 7(b) is converted into Delaunay mesh, as seen in Figure 7(c). Delaunay mesh is generated by removing the illegal internal edges iteratively, using edge swaps[23]. CA points exist at the midpoint of the internal edges and at the circum-center of the acute angle triangles, which have three internal edges. Centroid is used instead of circum-center to reduce the computational cost. The radius of CA points is calculated using half the length of internal edges and average distance from centroid to three vertices of acute angle triangle. For the purpose of mesh sizing, only a subset of skeleton points is considered. As the radius of skeleton points formed by adjacent curves approaches zero (see Figure 5(a) and Figure 7(c)) at convex vertices, these points will generate a fine mesh and hence the skeleton is trimmed. If the angle between the curve normal measured at the end vertices of the internal edge connecting the adjacent curves, is less than a threshold value, then the skeleton point at the mid-point of the internal edge is not generated. The size at these regions is determined by other tools discussed in the rest of this section and interpolation technique, which is detailed in Section 9.



(a) Graphics facets (b) Subdivided graphics facets

Figure 7 Skeleton generated in T_{pln}

(c) Trimmed disconnected CAT

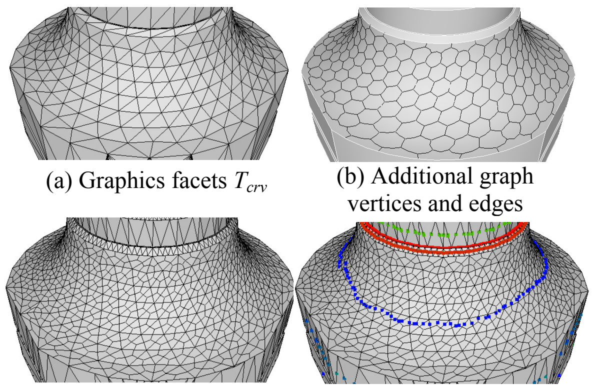
The disconnected skeleton generated in T_{crv} by propagating grassfire from the boundary curves is explained here, which is similar to 3D discrete skeleton generation[24]. Figure 8 (a) shows the initial graph T_{crv}, lying on a curved surface F. To improve the accuracy of skeleton, additional graph edges and vertices (see Figure 8(b)) are added to T_{crv} to form a dense graph T'_{crv}, as shown in Figure 8(c) and Figure 17(b). These additional edges are obtained by joining the centroid of the facets with the three midpoints of its edges. The accuracy of the skeleton can be improved by using multiple refinements, at the expense of time. The grassfire propagation on graph T'_{crv} has three main steps:

(1) The initiation phase, (2) propagation phase, and (3) termination phase (see Algorithm 1), which are detailed below.

In the initiation phase, the vertices of T'_{crv} lying on the boundary curves of F are inserted into a priority queue, H, to form the initial front for grassfire propagation. The distance, d, traveled by the wave at the vertices of the initial front is set to 0.0 and the curve ID is stored. The inward direction of the wave, **w**, at each vertex of the initial front is given by **n x t**, where **n** is surface normal and **t** is curve tangent. The wave direction, **w**, lies on the tangent plane of the surface and is orthogonal to **t**. In the propagation phase, the initial front is incrementally moved inwards by popping the top vertex, v_{top}, from H and pushing valid adjacent vertices of v_{top} into H. At the top vertex v_{top}, the distance d traveled by the wave is least. The valid adjacent vertices of v_{top} are the ones, which are not yet visited by the wave and whose insertion into the current front (H) will move the front inward. As the grassfire propagates, curve ID, distance, and direction of the wave at the newly-inserted adjacent vertices v_{adj} are calculated using v_{top} (performed in Function VisitNode in Algorithm 1). The distance d_{adj} at v_{adj} is given by $d_{adj} = d_{top} + \text{dist}(v_{top}, v_{adj})$. As the wave touches v_{adj} from different direction, the d_{adj} is updated by retaining the minimum value. Thus every node will have minimum distance from the boundary. The wave direction at the v_{adj} is calculated by taking the projection of wave direction at v_{top} on the tangent plane at v_{adj}. In the termination phase, skeleton points are generated at the region where the opposing fronts meet. As previously mentioned, the curve ID and the direction of the wave at the opposing fronts are used to trim the skeleton to avoid unnecessary fine mesh at convex corners. The position, distance and direction of the wave stored at the v_{top}, and its opposing adjacent vertex, is used to determine the position and radius of skeleton points. The grassfire propagation ends when the number of vertices in the current front H goes to zero.

```
Algorithm 1: Grassfire propagation on $T'_{crv}$
Input: $T'_{crv}$
Output: Disconnected skeleton points
Begin
  Insert boundary vertices of $T'_{crv}$ into a priority queue $H$.
  While ($|H| \neq 0$)
    $v_{top} \leftarrow H$.Pop()
    VisitNode ($v_{top}$, $H$)
End
```

Note that an approximate intrinsic distance, i.e., distance measured on the surface, is used, rather than Euclidean distance, in determining the radius of the skeleton points in both T_{pln} and T_{crv}. Also, since many of the meshing algorithms use the advancing front method (AFM)[1], the radius function of the skeleton is naturally an accurate estimation of proximity. Thus, computationally efficient, sufficiently accurate skeleton points are generated to measure the proximity in *in(F)* for the purpose of mesh sizing.


(a) Graphics facets T_{crv} (b) Additional graph vertices and edges

(c) Dense graph T'_{crv} (d) Disconnected skeleton

Figure 8 Skeleton generated in T'_{crv}

5.3. Measuring Curviness in *in(F)*

Unlike proximity, measuring curviness is a local problem. Let us consider a point $\mathbf{q} = \mathbf{x}(u_0, v_0)$, on surface F and let $\alpha(t)$ be a curve passing through **q** (see Figure 9(a)). $\alpha'(t)$ denotes the tangent vector of the curve α at t. The tangent plane of F at **q** is the set of tangent vectors of all curves passing through **q** and is given by

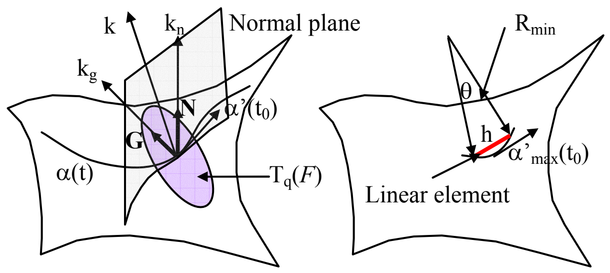
$$T_q(F) = \{\, \mathbf{v} \mid \mathbf{v} \text{ is tangent to } F \text{ at } \mathbf{q} \}.$$

The curviness of F at **q** can be measured by measuring the curvature of curve $\alpha(t)$ at **q**. Without any loss of generality, let $\mathbf{q} = \alpha(t_0)$, $\mathbf{w} \in T_q(F)$ be equal to $\alpha'(t_0)$, and **N** be the normal to $T_q(F)$ at **q**. The curvature vector **k** of α at **q** is given by Equation 6.

$$\mathbf{k} = \mathbf{k_n} + \mathbf{k_g} = k_n\,\mathbf{N} + k_g\,\mathbf{G} \qquad \textbf{(6)}$$

where **G** is the unit vector along $(\mathbf{N} \times \mathbf{w}) \in T_q(F)$, as shown in Figure 9. $\mathbf{k_n}$ is called “the normal curvature vector”, which is given by Equation 7; and $\mathbf{k_g}$ is known as “the geodesic curvature vector”. Their signed lengths, k_n, k_g are called normal curvature and geodesic curvature in direction of $\alpha'(t_0)$ respectively. Note that the signs of k_n and k_g depend on the orientation of the surface given by the normal vector, **N**.

For mesh sizing purposes, we are interested in the deviation in the direction of the normal while measuring the amount of curviness, which is clear from Figure 4(b). k_n is a measure of change in $\alpha'(t_0)$ in the normal plane, whereas k_g is a measure of change in $\alpha'(t_0)$ in the tangent plane. As the normal curvature k_n varies with the direction of $\alpha'(t_0)$, there exist extreme values k_{min} and k_{max} (not necessarily distinct values) [25], called “minimum and maximum principal curvatures”, along $\alpha'_{min}(t_0)$ and $\alpha'_{max}(t_0)$, respectively. Their product $K := k_{min} \cdot k_{max}$ is called “the Gaussian curvature” and their mean $H := (k_{min} + k_{max}) / 2.0$ is called “ the mean curvature”.



(a) normal and geodesic curvature (b) mesh size

Figure 9 Curvature and mesh size at a point

$$\mathbf{k}_n = \left(\frac{d^2\mathbf{x}}{dt^2} \cdot \mathbf{N} \right) \mathbf{N} \quad \textbf{(7)}$$

For the purpose of mesh sizing, it is necessary to evaluate the curvature at just a few sample points where the surface bends, at less computational cost. Mclvor et al. [26] compared eight different methods of estimating surface curvature. The facet-based method proves efficient for calculating a relatively accurate curvature estimate with less computational time. Sample points are placed where the direction of the facet's normal change in order to capture surface curvature (see Figure 10). As internal facet vertices exist to capture the curvature, sample points are placed on every internal facet vertices, as shown on the left in Figure 10. If the adjacent facets of an internal edge are not coplanar, then sample points are placed at midpoint of the internal edge and at centroid of adjacent facets, as shown on the right in Figure 10.

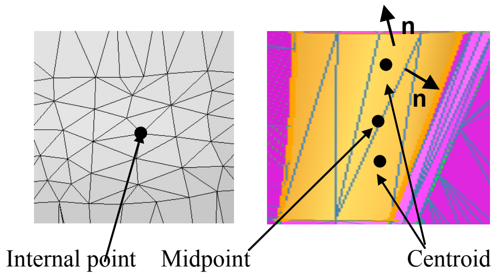


Figure 10 Sample points on graphics facets

5.4. Measuring Proximity in *in(C)*

On a curve, the midpoint acts as a skeleton point, and half the length of the curve is used to measure proximity between the end points. The length of curve α(t) between the end vertices α(a) and α(b) is given by Equation 8.

$$L(\alpha) = \int_a^b |\alpha'(t)| dt \quad \textbf{(8)}$$

As before, in order to reduce the computational cost, the length is calculated by adding the length of facet edges connecting the two end points of the curve α(t).

5.5. Measuring Curviness in *in(C)*

For a curve α(t) whose tangent vector is given by α'(t), curvature vector **k** is given by Equation 9. Curvature **k** measures the variation in the direction of the tangent vector. Sample points are placed on the curve at the facet vertices only if the adjacent facet edges are not collinear. The approximate curvature can be estimated by using the normal of adjacent edges to reduce the computational cost.

$$\mathbf{k} = \frac{|\alpha' \times \alpha''|}{|\alpha'|^3} \quad \textbf{(9)}$$

5.6. Measuring Twist in *in(C)*

Torsion is used to measure the twist of the curve α(t) and is given in Equation 10.

$$\tau = \frac{(\alpha' \times \alpha'') \cdot \alpha'''}{|\alpha' \times \alpha''|^2} \quad \textbf{(10)}$$

6. COMPUTATIONAL PROCEDURE FOR GENERATING FE MESH SIZING

The computational procedure for generating the mesh sizing function has three main steps: (1) generation of source points, (2) generation of a lattice for storing the sizing function, and (3) interpolation of mesh size on the lattice using source points. These three steps are discussed in the following paragraphs. The input is a set of connected surfaces. The boundary of the industrial mechanical parts is one example of the input (see Figure 16 and Figure 17). Here, the input surface to be represented is considered in B-rep format (such as the ACIS sat file), as it is supported by many commercial geometric modelers.

Although only geometric factors are considered in generating the sizing function here, the computational procedure proposed is sufficiently flexible to consider non-geometric factors, such as physics, boundary condition etc. The tools used to measure the geometric complexity of the input surface introduce different types of source points on the surface (see Figure 16(b) to Figure 16(d), more details in Section 7).

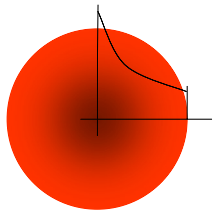
To store the mesh sizing function, a specific type of hierarchical background grid called PR-Octree (PR:Point Region) [27], is proposed (detailed in Section 8). In the literature, 2D background mesh [7] and 2D background grid [11] in parametric space were used to store mesh size. This approach depends significantly on the parameterization used. It is also expensive to find the triangle containing the point, where mesh size must be evaluated. Therefore a hierarchical background grid is generated here in 3D space, avoiding problems related to parameterization and reducing the meshing time.

The next step is to interpolate the sizing function over the PR-Octree lattice using the source points. A good interpolation scheme should provide flexibility in controlling mesh size and gradation [28, 29]; and it should generate smooth sizing function by satisfying *α-Lipschitz* condition. A good interpolation scheme should be computationally efficient, and thus able to generate variety of meshes as per user requirements. The details of the proposed interpolation scheme, which meet the above requirements, are explained in Section 9.

This computational procedure is independent of the meshing algorithm used and during mesh generation, the target mesh size at a point is interpolated using the size stored at the lattice-nodes of the octree cell containing that point.

7. GENERATION OF SOURCE POINTS

"The source point" is an abstract term which represents size and gradient at a location due to any of the factors, such as geometric, physics, loading, user defined, etc. Figure 11 shows a source point that is defined by: size, *s*; center, **c**[x,y,z]; scope, *scp*; and local sizing function , *f*. The size at the center **c** is denoted by *s,* and the sizing gradient around the center, inside the scope *scp,* is controlled by the local-sizing function *f*. By controlling the size, scope, and local sizing function, a variety of meshes can be generated to meet the user requirement.



s = Radius of skeleton point
c = Position of skeleton point
scp = Radius of skeleton point

$$f(\mathrm{r}) = s - \frac{\mathrm{r}}{scp} \times \left(s \times end\_factor\right)$$

General source point Skeleton-based source point

Figure 11 General and skeleton-based source points

Here, source points are generated using the tools (see Section 5) that measure geometric complexity of a surface. The disconnected skeleton is used to generate the skeleton-based source points. The size, center, scope, and local sizing function *f* , of the skeleton-based source points are determined as given in Figure 11. A linear local sizing function has been used. The *end_factor* controls the size at the periphery of the source points. By default, *end_factor* is set to 0.1, thus the size decreases with the distance away from the center. This will generate slightly smaller elements at the boundary; the intention being that FE mesh will better represent the geometric model at the boundary.

The surface-curvature-based source points are generated only at the sample points where the curvature is estimated (see Section 5.3). At a sample point **q ,** the element size 'h' is calculated as given in Equation 11 [30], (see Figure 9(b)), where θ is the maximum spanning angle and minimum radius of curvature ($R_{min} = 1/ k_{max}$) as seen in Figure 9(b). For a constant curvature, the deviation of the linear FE from the surface will be R_{min} - R_{min} . cos(θ/2). Mesh size h, shown in Figure 9(b), is a conservative estimate, as k_{max} is used in estimating the radius of curvature. Instead of k_{max}, mean curvature H can be used, which will result in coarser mesh size. Researchers have also used the combination of G and H [31]. The scope of the source points can be used to control the influence of curvature.

$$h = 2 \cdot R_{\min} \cdot \sin\left(\frac{\theta}{2}\right) \qquad \textbf{(11)}$$

The size of the curve-length-based source points is determined using half the length of the curve; the points are placed along the length of the curve at equal intervals. Like surface-curvature-based source points, curve-curvature-based source points are added on the curves at the facet vertices only if the adjacent facet edges are not collinear.

Note that, size of all the above mentioned source points are truncated to d_{min} and d_{max}, if the size is outside the user specified limits d_{min} and d_{max} (see Section 3).

8. PR-OCTREE LATTICE GENERATION

PR-Octree is selected over other types of octree [27] for storing, because it provides a suitable lattice for storing the mesh sizing function. PR-Octree [27, 29] is one of the hierarchical data structures used for special decomposition which represents a set of input points. By definition, the PR-Octree contains cells (cubes) which are either empty, or which contain only one point. The shape of the PR-Octree is independent of the order in which the input points are inserted. One disadvantage of the PR-Octree is that maximum depth depends on the minimum distance between any two points; therefore maximum depth (max_depth) of the PR-Octree is fixed here and is taken as a user input. The max_depth can be increased if input models contain fine features and high curvature regions.

In Figure 12 PR-Octree generation is explained using a quadtree for clarity. The vertices and centroid of the graphics facets of the input surfaces are given as the input points. The graphics facets capture the surface and curve, curvature and small features. Small-sized facets exist at the high curvature region and at fine features (see Figure 16(a) and Figure 17(a)). Therefore the density of facet points will be higher at high curvature regions and at small features (see left-side image in Figure 12); thus small-sized octree cells are generated in these regions (see right side quadtree in Figure 12 and octree in Figure 16(e)).

Even though the PR-Octree captures fine features and surface curvature, it is not directly suitable for storing sizing function, because of the presence of larger cells in the interior (see left-side quadtree in Figure 12). The cells are further subdivided until only 1-level of depth difference is maintained between the adjacent cells; this ensures smooth transition in cell dimension (see the right-side quadtree in Figure 12). The octree lattice can be further subdivided, based on the source points, to adapt the lattice based on mesh sizing. Thus a suitable lattice for storing sizing function is generated.

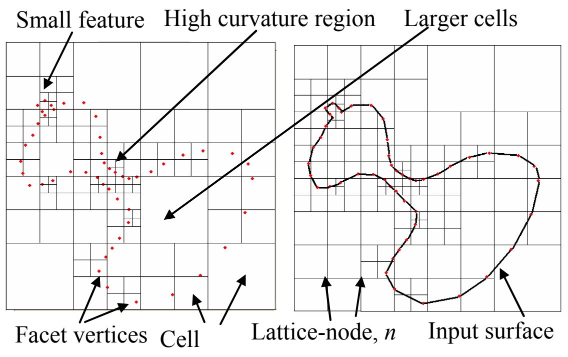


Figure 12 Original and Subdivided PR-Quadtree

9. INTERPOLATION USING SOURCE POINTS

In this section, the details of interpolating mesh size on the PR-Octree using the source points are described. The lattice-nodes of the octree cells that intersect with the graphics facets are marked as bnd-lattice-nodes. To limit the influence of a source point inside its scope, each source point is linked with the bnd-lattice-nodes that fall inside the scope of the source point. At first, the mesh size due to each type of source point is individually calculated at each bnd-lattice-node, by not merging with other source points, in order to preserve the characteristics, and to provide weights to each type source points. Later, the combined size at each bnd-lattice-node is calculated by taking the minimum of all the individual sizing functions, as shown in Figure 14. To ensure smooth gradient, i.e., to satisfy the *α-Lipschitz* condition, smoothing techniques are applied using digital filters. The following paragraphs explain this process in detail.

Size at a bnd-lattice-node '*n*', is interpolated by a particular type of source points, rather than by considering all types of source points at once. The size at *n* is interpolated by taking the weighted sum of the sizes determined by the local sizing function *f* of *m* source points of type k, linked to that lattice-node, as given in Equation 12 (see Figure 13). The weight at every source point is calculated by averaging the normalized weights, determined by the inverse square distance and inverse square size. This ensures that a source point which is nearer and has smaller size, has greater influence at a lattice-node

$$s_k = \sum_{i=1}^{i=m} f_i(d_i) \times \left(\frac{W_{i\_dist} + W_{i\_size}}{2} \right) \qquad \textbf{(12)}$$

where $$W_{i\_dist} = \frac{\frac{1}{d_i^2}}{\sum_{j=1}^{j=m} \frac{1}{d_j^2}}, \quad W_{i\_size} = \frac{\frac{1}{f_i(d_i)^2}}{\sum_{j=1}^{j=m} \frac{1}{f_j(d_j)^2}}$$

and d_i is the distance between the center of ith source point of type k and the lattice-node (*n*). k could be any type of source point discussed in Section 7. And $f_i(d_i)$ is the size determined by the local sizing function of the ith source point.

The final size 's' at each bnd-lattice-node is calculated as given in Equation 13, where 'i' represents different types of source points. *w* are the weights, and all weights are initially set to 1.0. Figure 14 shows the graphic view of Equation 13.

$$s = \max\{\min\{\min\{s_i \cdot w_i\} \cdot \text{overall\_scale}, d_{\max}\}, d_{\min}\} \qquad \textbf{(13)}$$

Note that individual sizing functions may not cover all the bnd-lattice-nodes, as some of these bnd-lattice-nodes may not be linked with any of the source points. The sizes at these bnd-lattice-nodes are calculated by averaging the non-zero size of adjacent bnd-lattice-nodes.

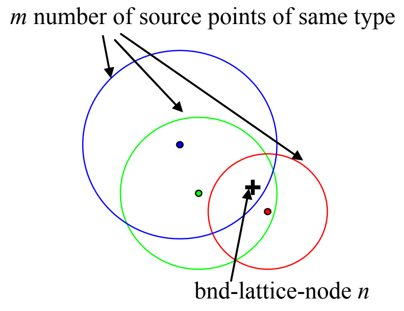


Figure 13 Interpolation at a lattice-node

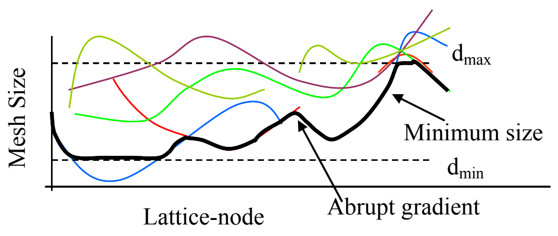


Figure 14 Minimum of all sizing functions

Smoothing techniques are used to alleviate the abrupt gradients caused by taking the minimum of all the individual sizing functions (see Figure 14). In the meshing literature, various smoothing techniques have been used as post-meshing techniques to smooth the FE meshes. Here smoothing is done before the meshing process, and thus a vector (position vector of the mesh nodes), is replaced by a scalar (mesh size). The concepts of digital filters, used in the image processing and pattern recognition in removing the noise, smoothing, and extraction of features, is used here in smoothing the mesh sizing function stored on the octree lattice. First the median filter is used to remove the outliers, then mean filters are used iteratively to smooth the gradients. By default, the number of iterations of mean filtering is set to two. While smoothing using mean filter, an additional constraint is used: if the average of the non-zero sizes of the adjacent lattice-nodes of a lattice-node *n* is more than the size at *n*, then the size at *n* is not altered, in order to respect the smaller size. This process is repeated iteratively until α-Lipschitz criterion is satisfied.

To find size at a point **p** on the surface *F,* during mesh generation, octree is first traversed from the root until the cell c_p , containing **p,** is found; then tri-linear interpolation is used to calculate the target mesh size at **p**, using lattice-nodes of c_p. Therefore the target mesh size is calculated in O(max_depth).

10. RESULTS AND DISCUSSION

The proposed approach has been implemented in C++ in CUBIT, the mesh-generator software of Sandia National Laboratories. The algorithm has been tested on many industrial models and results obtained on a few of these are shown in Figure 15 to Figure 17 (model courtesy of Ansys, Inc). The mesh sizing command takes the arguments, min_depth (int=4), max_depth (int=7), overall_scale (0.0 to 1.0 = 0.75), and interpolation_scheme (0 to 5=2). The numbers inside the parenthesis show the default values. From Table 1 is clear that total time of mesh sizing function generation (excluding meshing time) in Part 2 and Part 3--which contain 2154 and 3972 tris respectively-- is reasonable. The timings are measured in hp pavilion ze5155 notebook.

The disconnected skeleton generated using the proposed approach is computationally less expensive and accurate enough for mesh sizing purposes. Figure 16(b) and Figure 17(c) show the disconnected skeleton of Part 2 and Part 3. Table 2 shows that the skeleton is generated in a reasonable time of around 3 sec in both Part 2 and Part 3.

The overall_scale is used to control the coarseness level of the mesh. The overall_scale takes the value between 0.1 to 1.0; and overall_scale = 0.1 generates the finest mesh and overall_scale = 1.0 generates the coarsest mesh. Figure 15 shows the meshes generated with different values of overall_scale, but with the same bound values d_{max} and d_{min}.

By default, the interpolation_scheme is given by Equation 12 and by controlling the weights (see Equation 13), various types of meshes can be generated. A higher weight will increase the corresponding sizing function (see Figure 14). Figure 17 shows the meshes generated by changing the weight of curvature-based-source points.

Table 1 Computational time (sec) for generating mesh sizing function

	Part 2	Part 3
Octree Generation	8.31	14.70
Source Point Generatioin	5.84	3.51
Interpolation	4.57	2.29
Total	18.72	20.50

Table 2 Computational time for source point generation

No. of source points /Time(sec)

Source point type		
	Part 2	Part 3
Skeleton	5,377 / 3.01	4,053 / 3.17
Surface Curvature	3,723 / 0.64	1,462 / 0.13
Curve Curvature	287 / 0.15	492 / 0.10
Curve Length	492 / 0.01	318 / 0.01

11. CONCLUSION

In this paper, a computational procedure for FE mesh sizing function generation is proposed by systematically analyzing the geometric complexity of a set of surfaces. The tools that are sufficient to completely measure the complexity of surfaces are identified, and a reasonably accurate disconnected skeleton is generated using a new computationally-efficient algorithm for measuring proximity. The computational procedure uses PR-Octree lattice because it reduces the computational cost during meshing and the interpolation scheme is capable of generating a variety of meshes by controlling mesh size and gradation.

REFERENCES

[1] S. J. Owen, "A Survey of Unstructured Mesh Generation Technology," *Proceedings of 7th International Meshing Roundtable*, 1998.

[2] R. Lohner and P. Parikh, "Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method," *AIAA-88-0515*, 1988.

[3] L. P. Chew, "Constrained Delaunay Triangulations," *Algorithmica*, vol. 4, pp. 97-108, 1989.

[4] L. P. Chew, "Guaranteed-quality triangular meshes," *Tech. Report TR-89-983, Cornell University*, 1989.

[5] H. Chen and J. Bishop, "Delaunay Triangulation for Curved Surfaces," *6th International meshing roundtable*, 1997.

[6] A. Cunha, S. A. Canann, and S. Saigal, "Automatic Boundary Sizing For 2D and 3D Meshes," *AMD Trends in Unstructured Mesh Generation, ASME*, vol. 220, pp. 65-72, 1997.

[7] S. J. Owen and S. Saigal, "Neighborhood Based Element Sizing Control for Finite Element Surface Meshing," *Proceedings, 6th International Meshing Roundtable*, pp. 143-154, 1997.

[8] S. Pirzadeh, "Structured Background Grids for Generation of Unstructured Grids by Advancing-Front Method," *AIAA*, vol. 31, 1993.

[9] M. A. Yerry and M. S. Shepard, "A Modified-Quadtree Approach to Finite Element Mesh Generation," *IEEE Computer Graphics Applications*, vol. 3, pp. 39-46, 1983.
[10] W. C. Tracker, "A Brief Review of Techniques for Generating Irregular Computational Grids," *Int. Journal for Numerical Methods in Engineering*, vol. 15, pp. 1335-1341, 1980.
[11] M. S. Shephard, "Approaches to the Automatic Generation and Control of Finite Element Meshes," *Applied Mechanics Review*, vol. 41, pp. 169-185, 1988.
[12] H. Blum, "A Transformation for Extracting New Descriptors of Shape," *Models for the Perception of Speech and Visual Form Cambridge MA The MIT Press*, pp. 326-380, 1967.
[13] V. Srinivasan, L. R. Nackman, J. M. Tang, and S. N. Meshkat, "Automatic Mesh Generation using the Symmetric Axis Transformation of Polygonal Domains," *Proc. IEEE*, vol. 80(9), pp. 1485-1501, 1992.
[14] H. N. Gursoy, "Shape Interrogation by Medial Axis Transform for Automatd Analysis," *MIT Ph.D. Thesis*, 1989.
[15] H. N. Gursoy and N. M. Patrikalakis, "An Automatic Coarse And Fine Surface Mesh Generation Scheme Based on MAT Part I: Algorithms," *Engineering With Computers*, vol. 8, pp. 121-137, 1992.
[16] W. R. Quadros, K. Ramaswami, F. B. Prinz, and B. Gurumoorthy, "Automated Geometry Adaptive Quadrilateral Mesh Generation using MAT," *Proceedings of ASME DETC*, 2001.
[17] F.-E. Wolter, "Cut Locus and Medial Axis in Global Shape Interrogation and Representation," *Tech. Report, MIT Ocean Engineering Design Laboratory, revised version*, 1993.
[18] T. Rausch, F.-E. Wolter, and O. Sniehotta, "Computation of Medial Curves on Surfaces," *Welfen Laboratory Report No. 1*, 1996.
[19] F.-E. Wolter, "Local and Global Geometric Methods for Analysis Interrogation, Reconstruction, Modification and Design of Shape," *Computer Graphics International*, pp. 137, 2000.
[20] E. C. Sherbrooke, N. M. Patrikalakis, and F.-E. Wolter, "Differential and Topological Properties of Medial Axis Transforms," *Graphical Models and Image Processing*, vol. 58, pp. 574-592, 1996.
[21] M. D. Carmo, "Differential Geometry of Curves and Surfaces," *Prentice-Hall, Inc., Englewood, Cliffs, New Jersey*, 1976.
[22] L. Prasad, "Morphological Analysis of Shapes," http://cnls.lanl.gov/Highlights/1997-07/, 1997.
[23] M. d. Berg, M. v. Kreveld, M. Overmars, and O. Schwarzkopf, "Computational Geometry: algorithms and applications," *Springer*, 1997.
[24] W. R. Quadros, K. Shimada, and S. J. Owen, "3D Discrete Skeleton Generation by Wave Propagation on PR-Octree for Finite Element Mesh Sizing," *ACM Symposium on Solid Modeling and Applications*, 2004.
[25] J. Oprea, "Differential Geometry and its Applications," *Prentice-Hall, Inc., Upper Saddle River, New Jersey*, 1997.
[26] A. Mclvor and R. Valkernburg, "A comparison of local geometry estimation methods," *Machine Vision and Application*, vol. 10, pp. 17-26, 1997.
[27] H. Samet, "Spatial Data Structures," *in Modern Database Systems: The Object Model, Interoperability, and Beyond, W. Kim Ed. Addison-Wesley/ACM Press*, pp. 361-385, 1995.
[28] H. Borouchaki and F. Hecht, "Mesh Gradation Control," *6th International meshing roundtable*, 1997.
[29] P. J. Frey and P.-L. George, *Mesh Generation: Application to Finite Elements*, 1 ed: Hermes Science Publications, 2000.
[30] S. J. Owen and S. Saigal, "Surface Mesh Sizing Control," *International Journal for Numerical Methods in Engineering*, vol. 47, pp. 497-511, 2000.
[31] N. Dyn, K. Hormann, S. J. Kim, and D. Levin, "Optimizing 3D Triangulations using discrete curvature analysis," *Innovations in Applied Mathematics, Vanderbilt University Press*, 2001.

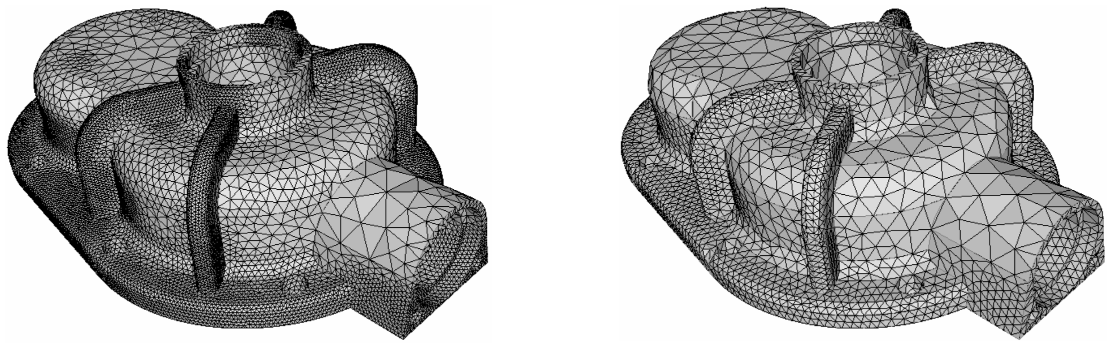
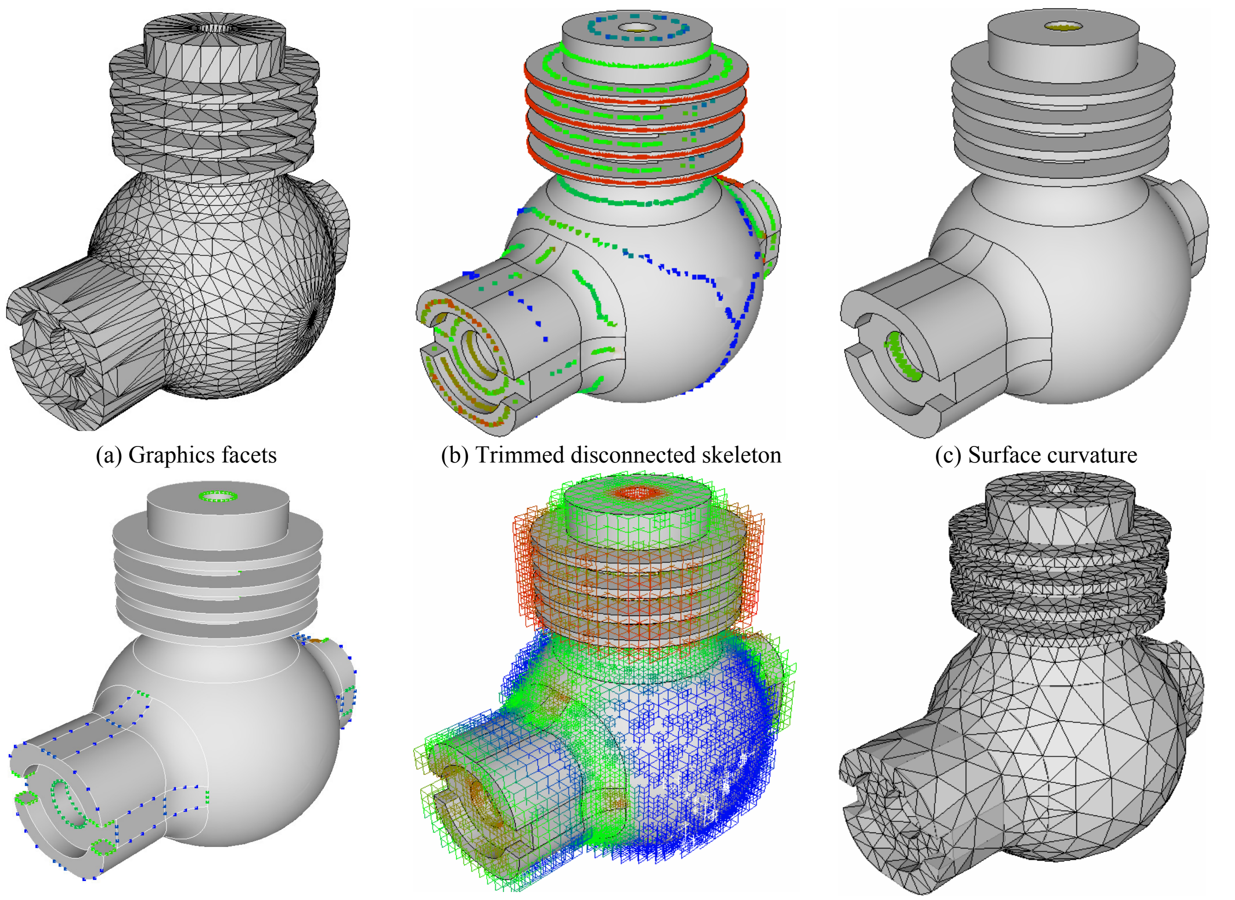
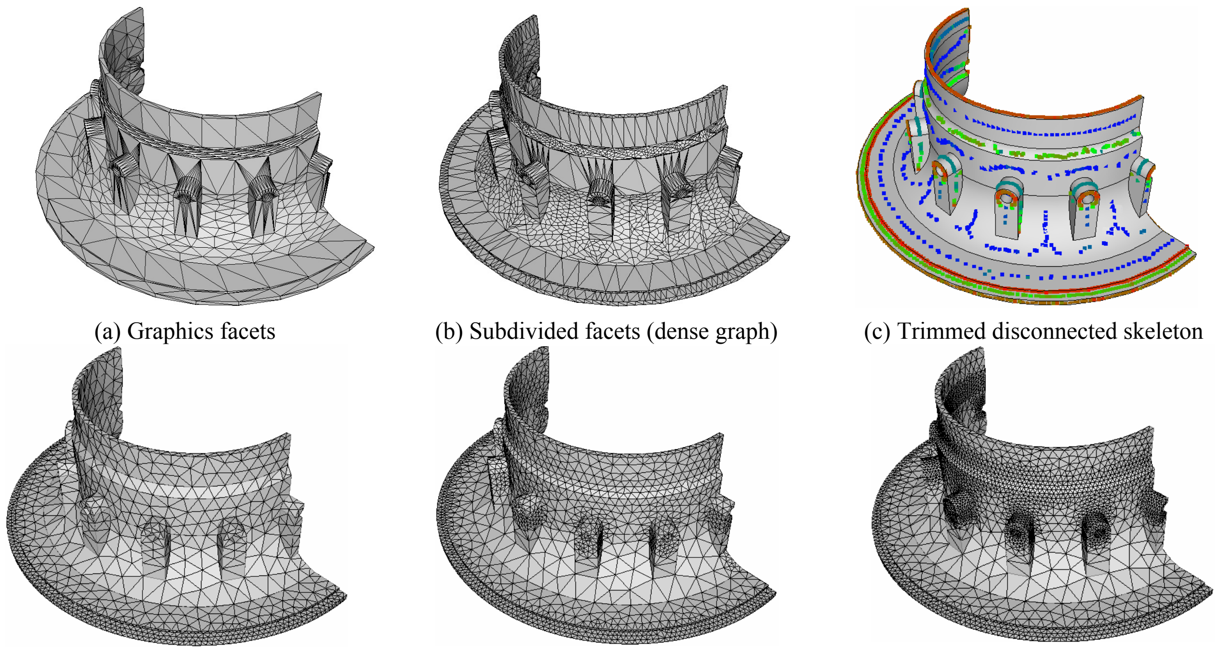

(a) overall_scale = 0.5, number of tris = 48,354 (b) overall_scale = 0.9, number of tris = 12,940

Figure 15 Effect of overall_scale on mesh size in Part 1


(a) Graphics facets (b) Trimmed disconnected skeleton (c) Surface curvature

(d) Curve curvature and length (e) Interpolation on PR-octree lattice (f) Geometry-adaptive mesh (2,154 tris)

Figure 16 Stages of mesh sizing function generation in Part 2


(a) Graphics facets (b) Subdivided facets (dense graph) (c) Trimmed disconnected skeleton

(d) curvature weight = 2.0 (e) curvature weight = 1.0 (f) curvature weight = 0.5

Figure 17 Effect of weight of curvature on mesh size in Part 3